CS 61C
Fall 2018

RISC-V Addressing and Caches

Discussion 5: September 24, 2018

1 RISC-V Addressing

We have several *addressing modes* to access memory (immediate not listed):

1. Base displacement addressing adds an immediate to a register value to create a memory address (used for lw, lb, sw, sb).

data acess

2. PC-relative addressing uses the PC and adds the immediate value of the in- struction (multiplied by 2) to create an address (used by branch and jump instructions).

control jumps/ mem accesses

3. Register Addressing uses the value in a register as a memory address (jr)

1.1 What is range of 32-bit instructions that can be reached from the current PC using a branch instruction?

12 bit imm $[-2^{11}, 2^{11}-1]$ halfwords

$1\ 00\ \dots\ 0\ 0$ 12   1 0

$[-2^{10}, 2^{10}-1]$ words (instructions)
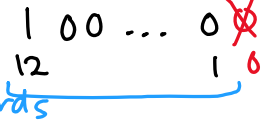
1.2 What is the range of 32-bit instructions that can be reached from the current PC using a jump instruction?
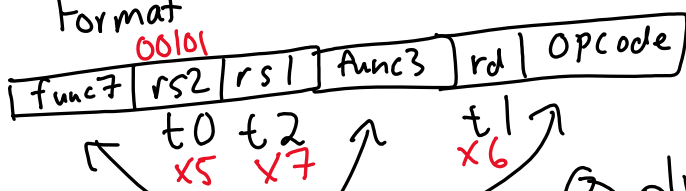
12 bit

1.3 Given the following RISC-V code (and instruction addresses), fill in the blank fields for the following instructions (you'll need your RISC-V green card!).

```
1  0x002cff00: loop: add t1, t2, t0
2  0x002cff04:       jal ra, foo
3  0x002cff08:       bne t1, zero, loop
4  ...
5  0x002cff2c: foo:  jr ra
```

					0x33
					0x6F
					0x63

ra=_____

① Find type of instruction

add : R type
jal : UJ-Type

② Find Format

func7	rs2	rs1	func3	rd	opcode

00101

t0 t2 ... t1
x5 x7 ... x6

Green Sheet

③ plug in values

jal

(3½)   0x002cff2c
      − 0x002cff04
      ─────────────
       0x 28 ⟹ 0b 0010 1000 = 00 ... 0 0010 100   0000
                                    20   (0:1)

log (Block Size) = offset bits

20 | 10:1   0000 0010 0100 0000
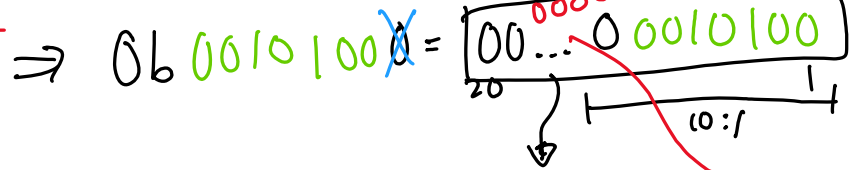
2   Understanding T/I/O

When working with caches, we have to be able to break down the memory addresses we work with to understand where they fit into our caches. There are three fields:

leftover

Tag - Used to distinguish different blocks that use the same index - Number of bits: leftovers

Index - The set that this piece of memory will be placed in - Number of bits: $\log_2(\#$ of indices)

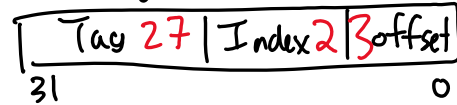**Offset** - The location of the byte in the block - Number of bits: $\log_2$(size of block)
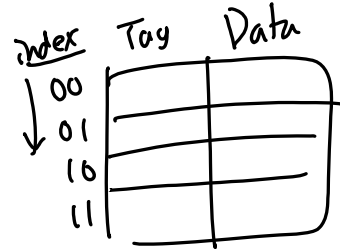
2.1 Assume we have a direct-mapped byte-addressed cache with capacity 32B and block size of 8B. Of the 32 bits in each address, which bits do we use to find the index of the cache to use?
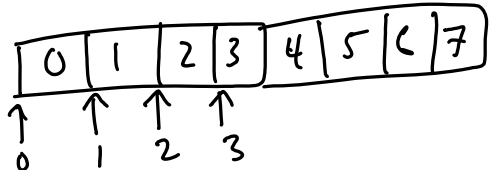
$$O: \log 8 = 3 \qquad \frac{32}{8} = 4 \qquad I: \log 4 = 2$$

2.2 Which bits are our tag bits? What about our offset?
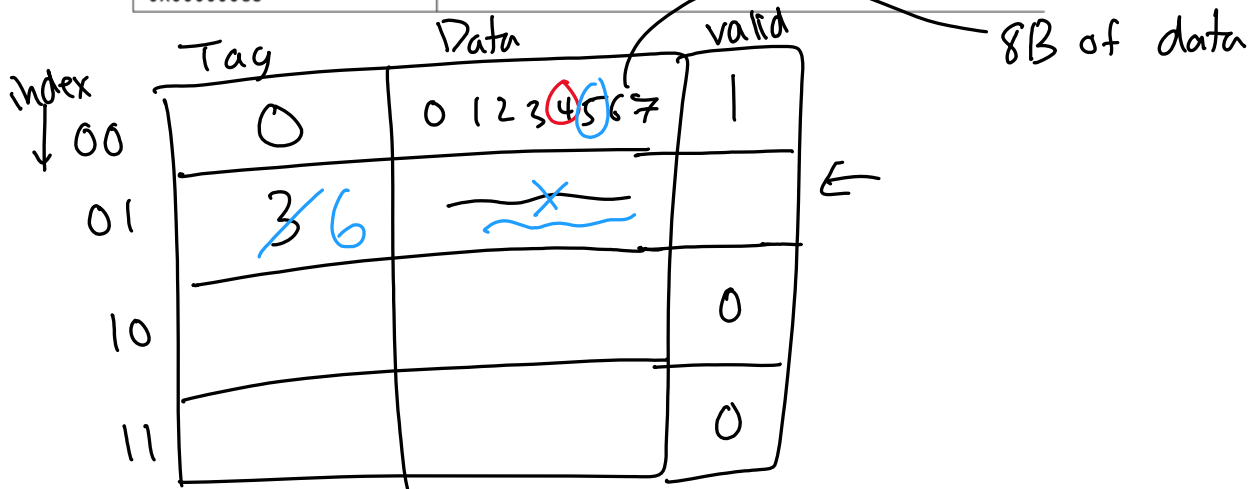
O $0:2$ I $3:4$ T $5:31$

2.3 Classify each of the following byte memory accesses as a cache hit (H), cache miss (M), or cache miss with replacement(R). It is probably best to try drawing out the cache before going through so that you can have an easier time seeing the replacements in the cache. The following white space is to do this:
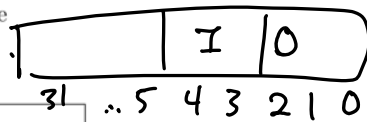
Address	T/I/O			Hit, Miss, Replace
0x00000004 .. 0100	0	0	4	M, comp
0x00000005 .. 0101	0	0	5	H
0x00000068 0110 1000	3		0	M comp
0x000000C8	6	1	0	M (R), Comp
0x00000068	3	1	0	R, conflict
0x000000DD				
0x00000045				
0x00000004				
0x000000C8				

index Tag Data
00
01
10
11

0 1 2 3 4 5 6 7
0 1 2 3

Tag 27 | Index 2 | 3 offset
31   0

I | O
31 .. 5 4 3 2 1 0

Tag   Data   valid   8B of data
index
00   O   0 1 2 3 4 5 6 7   1
01   3 6   ~~~x~~~
10   0
11   0

3   The 3 C's of Misses

3.1 Classify each M and R above as one of the 3 types of misses described below:

I. Compulsory: First time you ask the cache for a certain block. A miss that must occur when you first bring in a block. Reduce compulsory misses by having a longer cache lines (bigger blocks), which bring in the surrounding addresses along with our requested data. Can also pre-fetch blocks beforehand using a hardware prefetcher (a special circuit that tries to guess the next few blocks that you will want).

never see data before → comp miss

II. Conflict: Occurs if you hypothetically went through the ENTIRE string of accesses with a fully associative cache and wouldn't have missed for that specific access. Increasing the associativity or improving the replacement policy would remove the miss.

kick out could be better

III. Capacity: The only way to remove the miss is to increase the cache capacity, as even with a fully associative cache, we had to kick a block out at some point.

Cache too small

Note: There are many different ways of fixing misses. The name of the miss doesn't necessarily tell us the best way to reduce the number of misses.

4 ~~Extra~~ Practise

In the following diagrams, each blank box represents 1 byte (8 bits) of data. All of memory is byte addressed. Let's say we have a 8192KiB cache with an 128B block size how many bits are in tag, index, and offset? What parts of the address of 0xFEEDF00D fit into which sections?

0x0D = 0000 1101

	Tag	Index	Offset
Number of bits	9	16	7
Bits of address			000 1101

$$O: \log 128 = 7 \text{ bits} \quad I: \frac{2^{13} \cdot 2^{16} B}{2^7 B} = 2^{16} \quad \log 2^{16} = 16 \text{ bits}$$

$$32 - 16 - 7 = 9$$

4.2 Now fill in the table below. Assume that we have a write-through cache, so the number of bits per row includes only the cache data, the tag, and the valid bit.

Address size (bits)	Cache Size	Block Size	Tag Bits	Index Bits	Offset Bits	Bits per row
16	4KiB	4B				
32	32KiB	16B				
32			16	12		
64	2048KiB			14		1068